TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANIL SIRRI BASLAMISLI

CAMERA SENSOR INVARIANT AUTO WHITE BALANCE ALGORITHM WEIGHTING

Master of Science thesis

Examiner: Prof. Moncef Gabbouj
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 16 September 2015

ABSTRACT

ANIL SIRRI BASLAMISLI: Camera Sensor Invariant Auto White Balance Algorithm Weighting
Tampere University of Technology
Master of Science thesis, 42 pages, 0 Appendix pages
March 2016
Master's Degree Programme in Information Technology
Major: Data Engineering
Examiner: Prof. Moncef Gabbouj
Keywords: color constancy, illuminant estimation, auto white balance, machine learning, artificial neural networks, classification

Color constancy is the ability to perceive colors of objects, invariant to the color of the light source. The aim for color constancy algorithms is first to estimate the illuminant of the light source, and then correct the image so that the corrected image appears to be taken under a canonical light source. The task of the automatic white balance (AWB) is to do the same in digital cameras so that the images taken by a digital camera look as natural as possible. The main challenge rises due to the ill-posed nature of the problem, that is both the spectral distribution of the illuminant and the scene reflectance are unknown.

Most common methods used for addressing the AWB problem are based on low-level statistics assuming that illuminant information can be extracted from the image's spatial information. Nevertheless, in recent studies the problem has been approached with machine learning techniques quite often and they have been proved to be very useful.

In this thesis, we investigate learning color constancy using artificial neural networks (ANNs). Two different artificial neural network approaches are utilized to generate a new AWB algorithm by weighting some of the existing AWB algorithms. The first approach proves to be better than the existing approaches in terms of median error. On the other hand, the second method, which is better also from system design point of view, is superior to others including the first approach in terms of mean and median error. Furthermore, we also analyze camera sensor invariance by quantifying how much the performance of the ANNs degrade when the test sensor is different than the training sensor.

PREFACE

This thesis work has been conducted at the Department of Signal Processing of Tampere University of Technology in collaboration with Intel.

I would like to thank to my supervisor Academy Professor Moncef Gabbouj for providing me such an exciting subject and for the support throughout the thesis. Likewise, I would like to thank to Jarno Nikkanen and Intel for the opportunity and his continuous support and guidance.

Furthermore, I would like to thank to the members of the Multimedia Research Group (MUVIS) for their friendship and for warm working atmosphere. Especially, many thanks goes to Guanqun Cao for bringing different perspectives to the thesis and for his feedbacks.

It is not possible to mention everyone here, but I want to express my gratitude to Berkay Kicanaoglu and Hidir Yuzuguzel for their friendship, guidance, support and also for fruitful discussions from the beginning of the thesis until the end.

I would also like to expend my thanks to my beloved friends Begum Gokce Sagmaner, Cumhur Kusdemir and Ismailcan Ersahin for their continuous support, love and friendship. I remember the first time we met 7 years ago in Cyprus. Looking back to those days makes me realize that I am grateful that we have done everything together and grown up together. I would not have it any other way.

Finally, for their love and support, the warmest thanks goes to my family. I undoubtedly could not come this far without you. Thus, I would like to dedicate this thesis for them.

Tampere, 21.03.2016

Anil Sirri Baslamisli

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

AE	Auto exposure, page 1
AF	Auto focus, page 1
ANN	Artificial neural network, page 2
AWB	Auto white balance, page 1
BP	Backpropagation, page 17
CC	Color constancy, page 1
CNN	Convolutional neural networks, page 41
GD	Gradient descent, page 17
GW	Grey world, page 10
GWOC	Grey world one count, page 10
MLP	Multilayer perceptron, page 15
MSE	Mean squared error, page 17
RSS	Residual sum of squares, page 5
SGD	Stochastic gradient descent, page 17
SoG	Shades-of-grey, page 5
WGE	Weighted grey edge, page 5

$b^{(*)}$	all biases, page 16
f	image, page 9
G	von Kreis model, page 1
$intel\_*$	current Intel database, page 35
L	loss function, page 25
o	output of a perceptron, page 13
$O(x, y)$	output image, page 21
$psum_c$	Minkowski norm, page 11
sgn	sign function, page 13
w_i	a weight of a perceptron, page 13
y	weighted sum of inputs, page 13
$\triangle w_i$	weight change, page 13
$\triangle w_{kj}$	error gradient, page 18
$W^{(*)}$	all weights, page 16

1. INTRODUCTION

Computer vision aims to make computers understand images and video, or more precisely, the ability to see. In computer vision problems, we take visual data x and use them to infer something about the world w [25]. As a result, it deals with how we can combine sensor-derived images, world knowledge and knowledge of the image understanding to construct a model of the surrounding environment [11].

Computer vision problems are basically divided into 2 parts: low-level and high-level vision tasks. The latter includes many fields such as object recognition, motion analysis and 3D reconstruction, whereas the former includes noise reduction, sharpening, blurring and the famous 3A algorithms in a typical imagining pipeline of a digital camera: Auto Exposure (AE), Auto Focus (AF) and Auto White Balance (AWB). As explained in [12], AWB, also known as color constancy (CC), is the perceptual ability to distinguish the color of an object even under a colored light; in other words, we want to interpret the color of an object independently of the color of the light source. Both correct and incorrect white balance are illustrated in Figure 1.1. The main reason why color constancy is so challenging is that it is an *ill-posed inverse* problem, because both the spectral distribution of the illuminant and the scene reflectance are unknown.

Computational color constancy algorithms aim to solve the problem in two steps. Firstly, by using an AWB algorithm, they produce an estimate of the color of the light source of a target image based on the assumption that the color of the light source is uniform across the scene, then they correct the image so that the corrected image appears to be taken under a canonical light source [14]. As explained in detail in [22], the correction is achieved by using a diagonal model, known as von Kreis Model, of illumination change:

$$\vec{x'} = G \cdot \vec{x}, \qquad G = \begin{bmatrix} w'_R/w_R & 0 & 0 \\ 0 & w'_G/w_G & 0 \\ 0 & 0 & w'_B/w_B \end{bmatrix} \tag{1.1}$$

where $\vec{w} = [w_R, w_G, w_B]$ is the linearized camera response under one illumination and $\vec{w'} = [w'_R, w'_G, w'_B]$ is the response under another illumination. The diagonal model holds if given an image, we can convert all of its colors $\vec{x}$ from one illumination to another. The diagonal elements of G are also known as white balance gains.

Figure 1.1 *Correct vs. incorrect white balanced image*
(Source: https://photographylife.com/what-is-white-balance)

There are basically two types of illuminant estimation algorithms. The first type is static methods. They are based on low-level statistics assuming that illuminant information can be extracted from an image's spatial information. For example, grey world or white patch algorithms. The second type is learning-based methods, which requires a model to be trained and after to be used for the illuminant estimation. For instance, color by correlation or gamut mapping. Hence, unlike static methods, learning-based methods demand training images.

Auto white balance problem has been approached with machine learning techniques quite often. For example in [31] and [1], they applied different regression methods and in [7], [28] and [21] they addressed the problem by using neural networks. Neural networks have been shown to be feasible in solving auto white balance problem. In parallel with that, in this thesis, two different artificial neural network (ANN) approaches are utilized to generate a new AWB algorithm by weighting some of the existing AWB algorithms. The first approach proves to be better than the existing approaches in terms of median error. On the other hand, the second method, which is better also from system design point of view, is superior than all others including the first approach in terms of mean and median error.

The rest of the thesis is organized as follows. Chapter 2 discusses briefly the basics of

machine learning and auto white balance concept and algorithms. In Chapter 3, we present the pre-processing, image features and ANN models in detail. In Chapter 4 we present our experimental results. Finally, the thesis is concluded in Chapter 5.

2. THEORETICAL BACKGROUND

This chapter provides the basic knowledge for machine learning and auto white balance concepts that will be used throughout the thesis.

2.1 Machine Learning

Machine learning is one of the many areas in artificial intelligence. It is applied in cases where a computer cannot be explicitly programmed with a set of instructions to accomplish a task. Nonetheless, instead of having a set of instructions to solve a problem, sometimes we can have an idea or acceptable amount of data that can be used to construct a good and valuable approximation of the solution. The approximation can be achieved by learning an efficient model using the available data or past experience to make predictions. As a result, machine learning is about different models for learning and different methods to learn with the models by adapting their parameters from experience.Then, a computer program is said to learn if its performance P at some tasks T improves with their own experience E [20]. For example, for one of the most popular machine learning problems, hand writing recognition, T is recognizing and classifying handwritten words within images, P can be percent of words correctly classified and E can be a database of handwritten words with given classifications. Some of other well-known and real world machine learning examples also include face detection, customer segmentation and medical diagnostics.

There are two main subfields of machine learning; supervised learning and unsupervised learning. In supervised learning, algorithms are trained on labeled examples where the desired output is known. In other words, machine learning algorithms using supervised learning methods *learn* by inspecting a set of inputs and expected outputs to find out an efficient mapping function between them. There are several supervised learning approaches and algorithms in the literature, such as multilayer perceptron, decision tree, logistic regression, support vector machine, k-nearest neighbors algorithm and Naïve Bayes classifier. On the other hand, in unsupervised

learning, algorithms operate on unlabeled examples where the desired output is unknown. The purpose is to find interesting and potentially useful patterns and structure within data by using the probability densities. There exist some unsupervised learning techniques, yet the most widely used one is cluster analysis.

The two essential problems in supervised learning are regression and classification. An example application for classification would be classifying emails as 'spam' or 'not-spam'. A regression example would be estimating the price of a house. The most simple regression model is linear regression. Suppose we have data of the form $(x_1, y_1), ..., (x_n, y_n)$, where n denotes the sample size and x_i and y_i are real numbers with i in $\{1, ..., n\}$, the model predicts a quantitative response $f(X) = Y$ by assuming that there is approximately a linear relation between input X and output Y. Then, the simple linear regression model has the form:

$$Y = \beta_0 + \beta_1 X \tag{2.1}$$

In the model, β_0 and β_1 are two unknown parameters. By training the model, we want to estimate those parameters so that the linear model fits the available data well and is a good approximation. The estimation is done by minimizing a loss function. The loss function L is the sum of individual losses over the instances of X. Let $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ be the prediction for Y based on the i^{th} observation of X. Then, $e_i = y_i - \hat{y}_i$ represents the i^{th} residual, where y_i is the response variable. Then the loss function can be defined as the sum of residuals:

$$L = \sum_{i=1}^{n} e_i \tag{2.2}$$

We aim to choose β_0 and β_1 to minimize the L. Therefore, one of the key issues regression problems differ from each other is the choice of the loss function. The most common and simplest one is residual sum of squares (RSS):

$$RSS = \sum_{i=1}^{n} e_i^2 \tag{2.3}$$

On the other hand, classification is used for predicting qualitative responses. The goal is to take an input vector and assign it to one of the available classes by constructing a model based on the training set and the class labels. Normally, the classes are disjoint; thus, each input vector is only assigned to one class only. As a result, the input space is separated into decision regions with boundaries called *decision boundaries* or *decision surfaces* [5]. They are illustrated with green lines in Figure 2.1.

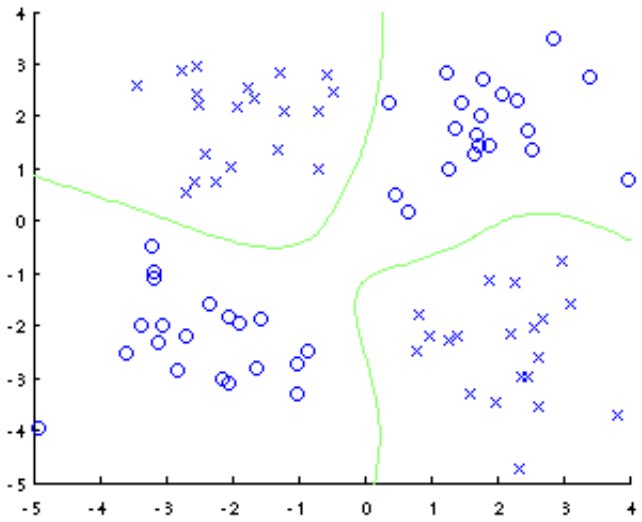

Figure 2.1 *Decision boundaries*
(Source: `http://www.work.caltech.edu/~htlin/program/libsvm/` *)*

A simple classification system includes several phases. The first one is *preprocessing*, which handles raw data obtained from a source to be properly utilized as an input, because raw data can be incomplete, noisy or inconsistent. To that end, this phase involves data related tasks, such as cleaning, integration, transformation, reduction or discretization. The next phase is *feature extraction*. Features are domain specific measurements containing relevant information in order to generate the best possible representation of an input. To illustrate, in a fruit classification task, a feature vector may include information about a fruit's size, color, shape, etc. For classification tasks we aim to generate feature vector representations with the highest intra-class and the lowest inter-class similarity. The third phase is *training*, which was explained earlier to build a model for prediction. Lastly, in the *classification* phase the system assigns the input to one of the available classes according to a decision rule.

As stated in [18], a classification problem has three fundamental parts. Firstly, the relative frequency of the classes to occur in the population; i.e. the *prior* probability distribution. Secondly, an implicit or explicit criterion that defines the mapping between input/output relationship for separating the classes. Finally, a loss function to penalize a wrong classification so that the total cost of misclassifications should be minimized as is in the regression case. Thus, most of the classification tasks based on the probability theory. A probabilistic model outputs a vector containing the probability distribution over all classes that shows the degree of certainty of each class.

After finalizing the training of a model and estimating the optimum parameters, we can use the model for unseen data to make predictions. However, if we over-train the model on the training data or use a model which is too complex, then it may only learn and memorize the training set and will perform quite poorly when presented with new data, which is called *overfitting*. Nonetheless, we want to create models that are capable of *generalizing* so that the model can perform well on new unseen data. The situation is illustrated in Figure 2.2. In the figure, from left to right the models having different complexities have been trained on the training data. The training error curve in the bottom box shows that it gets better and better as the complexity increases. On the other hand, prediction error for new data increases, as well. It can be observed from the top right graph the most complex model 'memorized' the training data, yet fails when new unseen data is presented. However, the graph on the middle seems to follow a pattern instead of going through each data point with less prediction error for new data. As a result, to create a good model capable of generalizing, we should be careful about the complexity of the model and when to stop training so that it does not overfit.

After generalizing the model, we should check for the performance assessment, which is an essential aspect of machine learning to determine how well a model can perform. There are several performance evaluation methods exist in the literature, such as resubstitution error rate or holdout error rate, yet the most common approach is called cross validation, also known as K-fold cross validation, which is also utilized in this thesis. In cross validation, first of all, a random permutation of the sample set is generated, then we create a K-fold partition of the shuffled dataset. For each of K experiments, we use (K-1) folds for training and the remaining fold for testing. The process takes place K times and we calculate the test error estimate as an average of test errors for each fold. An example case of 5-fold cross validation is illustrated in

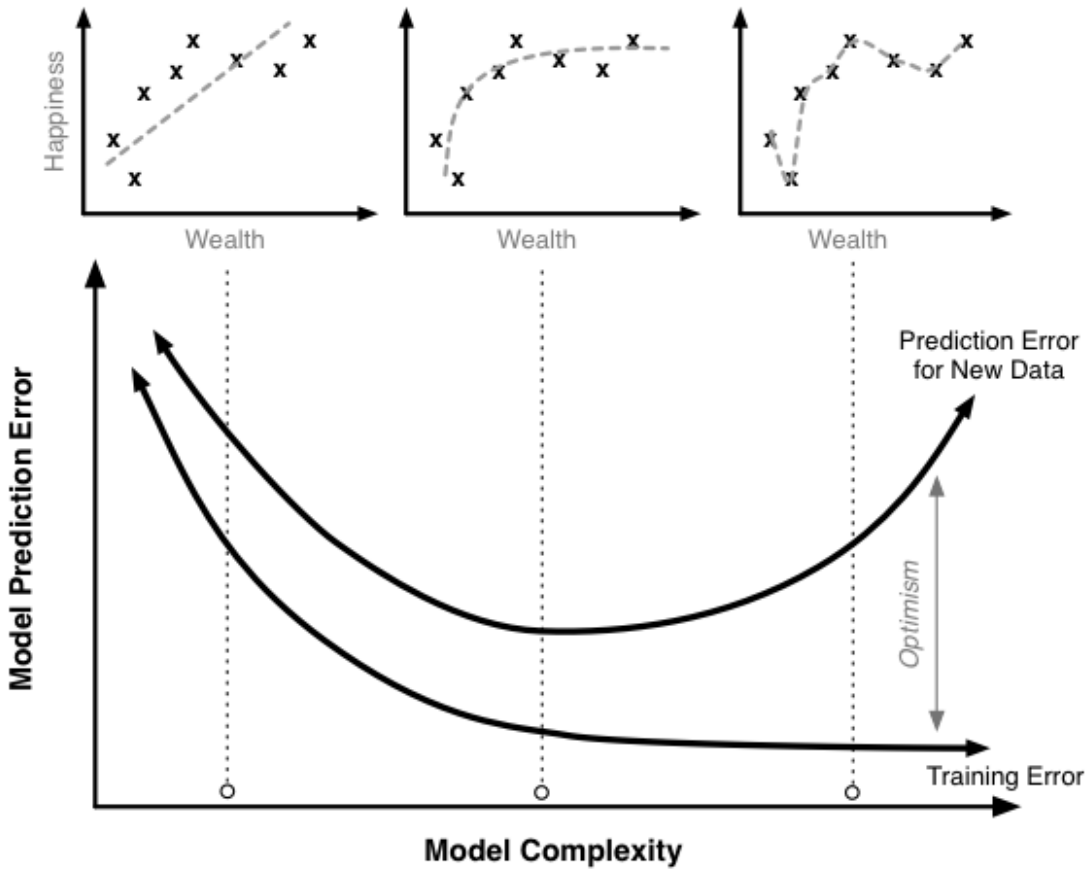

***Figure** 2.2 Overfitting example*
(Source: `http://scott.fortmann-roe.com/docs/MeasuringError.html`)

Figure 2.3. The process can be computationally expensive as it requires the design of K classifiers. Nonetheless, it is generally unbiased [30]. If we perform cross validation with K is equal to the number of samples, the process is called leave-one-out cross validation, which is considerably useful with small datasets, because K-fold cross validation demands larger sample sizes. In practical applications K = 5 and K = 10 are usual choices.

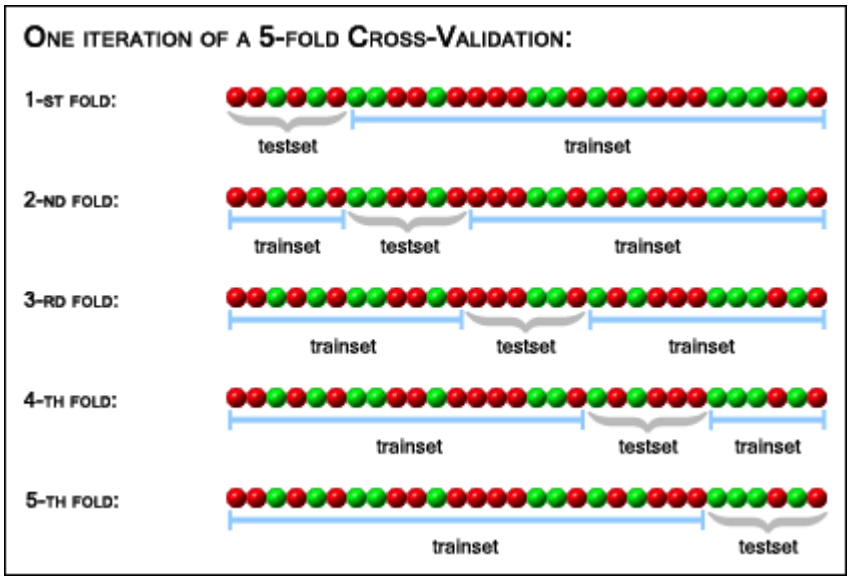

Figure 2.3 *5-Fold cross validation*
(Source: `http://genome.tugraz.at/proclassify/help/pages/XV.html` *)*

2.2 Color Constancy

As explained in detail in [29] and [15], the image values $f = (R, G, B)^T$ for a Lambertian surface depend on the color of the light source $e(\lambda)$ where λ is the wavelength, the surface reflectance $s(\lambda)$ and the camera sensitivity functions $c(\lambda) = (R(\lambda), G(\lambda), B(\lambda))$:

$$f = \int_{\omega} e(\lambda)s(\lambda)c(\lambda)d\lambda \tag{2.4}$$

where ω is the visible spectrum. With the assumption of the illumination of the scene is uniform, the aim of color constancy algorithms is to estimate the color of the light source $e(\lambda)$:

$$e = \begin{pmatrix} R_e \\ G_e \\ B_e \end{pmatrix} = \int_{\omega} e(\lambda)c(\lambda)d\lambda \tag{2.5}$$

2.2.1 Color Constancy Algorithms

In this thesis we use four different static illuminant estimation methods, namely grey world, grey world one count, shades-of-grey and weighted grey edge as our reference.

Grey World Algorithm (GW)

The algorithm assumes that the average reflectance of the surfaces in an image scene is achromatic [6]. In other words, the average pixel values of red, green and blue components of an image are equivalent to each other:

$$\frac{\int s(\lambda, x)dx}{\int dx} = k \tag{2.6}$$

where k is a multiplicative constant chosen such that the illuminant color e has unit length. As a result, the color of the light source can be estimated by computing the average pixel value of an achromatic scene, where the reflected color gives the color of the light source:

$$\frac{\int f(x)dx}{\int dx} = \frac{1}{\int dx} \int \int_{\omega} e(\lambda)s(\lambda, x)c(\lambda)d\lambda dx \tag{2.7}$$

$$= k \int_{\omega} e(\lambda)c(\lambda)d\lambda = ke \tag{2.8}$$

given the values $f(x)$, where x is the spatial coordinate in the image. Furthermore, since the algorithm is simple to implement and it has a very low computational cost, it is vastly used for benchmarking. However, it tends to fail when the image scene contains large colored areas, when the average scene is not grey, which leads to a color cast. For example, a large red object would cause color error towards cyan. Such an error is illustrated in Figure 2.4.

Grey World One Count Algorithm (GWOC)

The proposed algorithm is yet another version of the GW algorithm. However, the algorithm counts each separate chromaticity only once. In other words, in $[R/G, B/G]$ chromaticity plane, each $[R/G, B/G]$ that is present in the image frame is counted only once. Hence, the algorithm performs better when the GW algorithm tends to fail heavily where the large colored objects are presented in the field-of-view.

Figure 2.4 *An example showing the failure of the Grey World algorithm*

Shades-of-Grey Algorithm (SoG)

Shades-of-grey algorithm is more general form of GW algorithm as it assigns different weights to good and bad sides of GW algorithm assuming that the scene average is shades of grey. As proved in [10], GW algorithm is a specific case of a Minkowski norm given in below equation, where m and n are the dimensions of the image I:

$$\forall c \in \{R, G, B\}, I_c(x, y) : psum_c = \sqrt[p]{\frac{1}{mn} \sum I_c^p(x, y)} \tag{2.9}$$

For $p = 1$ the equation is equal to the GW algorithm assumption, for $p = \infty$ it is equal to Max-RGB algorithm, which assumes that the maximum response in the components of an image arises from a white patch, the lightest patch to use as a white reference [29]. However, this algorithm is not in the scope of this thesis. SoG algorithm assumes the optimal illuminant estimate is between L_1 and L_∞. Although how to choose the p value depends on the dataset, in [10], it was stated that when $p = 6$, the algorithm is comparable to many advanced color constancy algorithms.

Weighted Grey Edge Algorithm (WGE)

As it was proposed in [29], Grey-Edge algorithm is based on the fact that the largest variation of the distribution of color derivatives is directed towards the light source and the algorithm assumes that the average edge difference in a scene is achromatic:

$$\frac{\int |s_x^\sigma(\lambda, x)| dx}{\int dx} = g(\lambda) = k \tag{2.10}$$

where the subscript x implies the spatial derivative at scale λ. Hence, the color of the light source can be estimated from the average color derivative in the image based on Equation 2.8

For WGE, the proposed algorithm in [15] aims to use the information that is based on photometric features of distinct edge types such as material edges, shadow or shading edges, specular edges and interreflection edges by implementing an iterative weighting strategy to gradually increase the accuracy of the estimation of the color of the light source and update edge weights accordingly:

$$(\int |\omega(\mathbf{f})^\kappa f_{c,x}(x)|^p dx)^{\frac{1}{p}} = ke_c \tag{2.11}$$

where $\omega(\mathbf{f})$ is a weighting function and κ is used for dictating the weights. Such weighting scheme's first iteration, second iteration and convergence is illustrated in Figure 2.5, where the weight maps are color coded, such that dark red indicates a high weight and dark blue indicates a low weight.

2.3 Artificial Neural Networks

An artificial neural network is a massively parallel distributed system consisting of many simple processing units called neurons to model the way the brain performs a particular task by acquiring knowledge from its environment through a learning process [16]. The learning process is achieved by adjusting the synaptic weights of the network. Artificial neural networks are useful to solve complex classification tasks, because they are very fast due to its massively parallel distributed structure and their ability to learn and generalize.

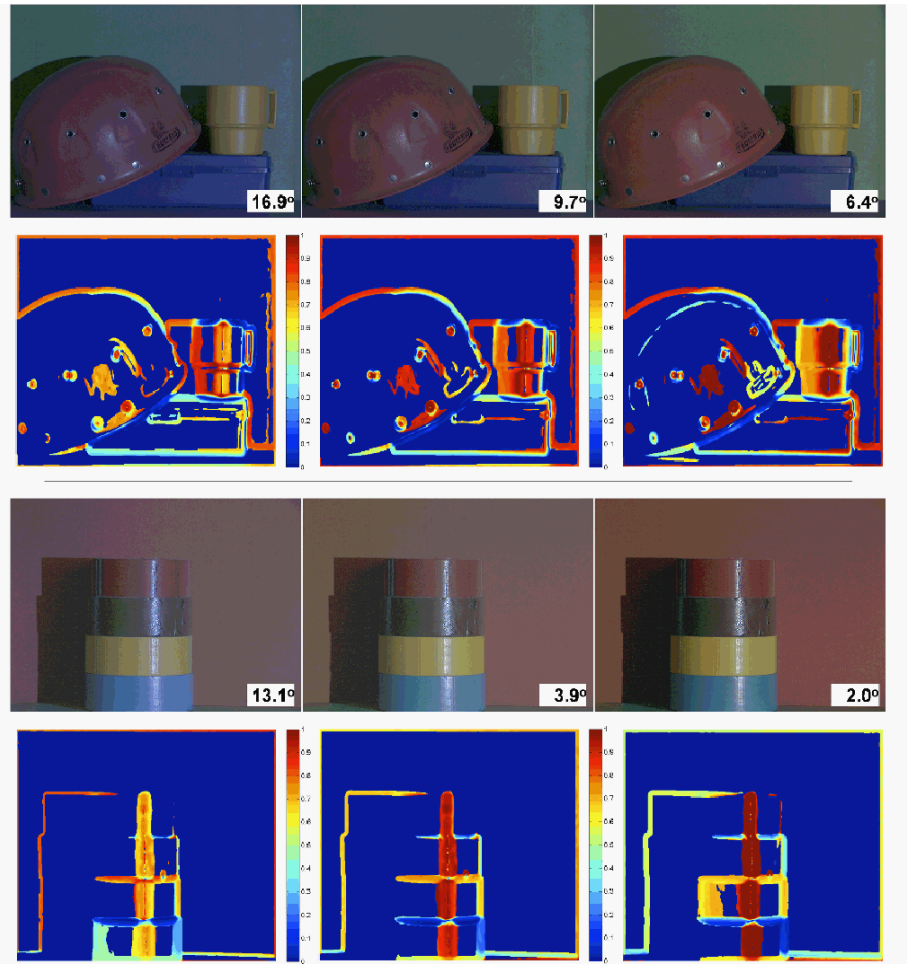

Figure 2.5 *WGE algorithm weight maps [15]*

2.3.1 Perceptron

A perceptron forms the basis of the ANNs. It is a feedforward network that builds linear decision boundaries. As illustrated in Figure 2.6, it has inputs represented by synaptic weights, an adder for summing the input signals and an activation function to determine the output of the system. In its simplest form, it can be defined as the weighted sum of its inputs:

$$y = \sum_{j=1}^{n} w_j x_j + w_0 \tag{2.12}$$

where n is the number of inputs, and w_0 is the intercept value coming from a bias unit $x_0 = +1$. The perceptron is activated by inputing of a pattern to its outputs. For linear cases, we can use a binary threshold activation function (e.g. step function)

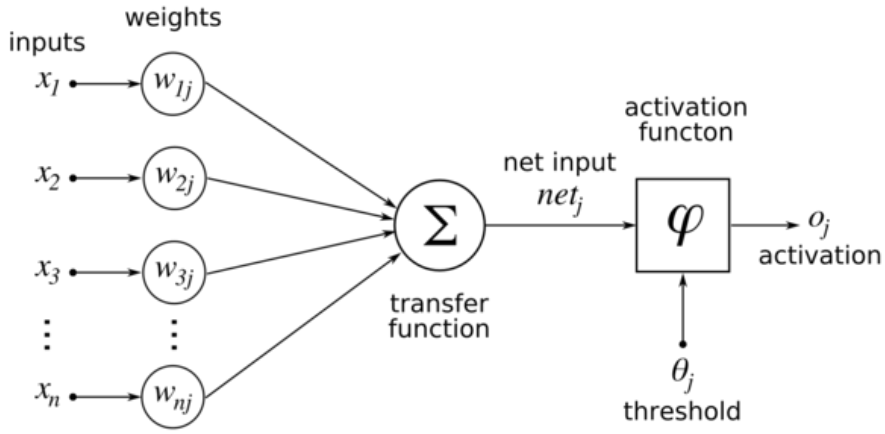

Figure 2.6 *A perceptron*
(Source: `https://nl.wikipedia.org/wiki/Perceptron`)

to determine the output. Then, we can decide the output of the system as:

$$o(x) = sgn(w \cdot x) \tag{2.13}$$

where

$$sgn(y) = \begin{cases} +1 & \text{if } y > 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.14}$$

On the other hand, to represent non-linear functions in general, we need non-linear activation functions such as *sigmoid* or *tanh*, which are illustrated in Figure 2.7. Sigmoid function takes real numbers and squashes them into range $[0, 1]$, where at 0 it is not activated at all and at 1 it reaches maximum frequency, whereas tanh function squashes real numbered numbers into range $[-1, 1]$:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.15}$$

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.16}$$

As a result, tanh can be considered as a scaled sigmoid function:

$$tanh(x) = 2sigmoid(2x) - 1 \tag{2.17}$$

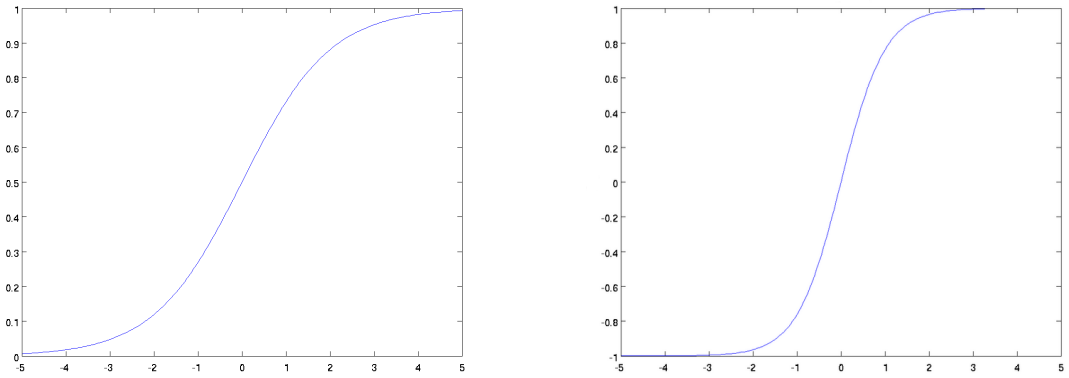

Figure 2.7 *Sigmoid vs. tanh*

Several algorithms can be used for the training of a perceptron to discover an optimum weight vector so that the output predicts correct class labels. The most common method is known as *the perceptron learning rule*, which always converges to optimum weights in finite time if the classification problem is linearly separable [26]. For this process, first we initialize the weights to small random values, generally between $[-1, +1]$. Then, perceptron is applied to each training example. If an output is correctly classified, we do not change the weights, otherwise weights are updated by adapting them at each iteration with:

$$w_i \leftarrow w_i + \triangle w_i \tag{2.18}$$

where $\triangle w_i$ denotes:

$$\triangle w_i = \alpha(t - o)x_i \tag{2.19}$$

In Equation 2.19 t is the target label of the current iteration's training example, o is the output of the perceptron, and α is a number generally between $[0, 1]$ called the *learning rate*, which controls how should the weights are updated at each iteration.

2.3.2 Multilayer Perceptron

Single perceptron is not able to solve complex tasks, because it is limited to linear mapping. As a result, we need a more generic model that is capable to perform arbitrary mappings. We can use perceptron to build a larger and more practical structure, called multilayer perceptron (MLP). Hence, an MLP can also be described

as a network of perceptrons. A typical MLP consists of an input layer, one or more hidden layers and an output layer. The hidden layer is visible to neither the inputs nor the outputs and allows the network to learn complex models by extracting useful features from the inputs. Such a network with one hidden layer, also known as two-layer perceptron, is shown in Figure 2.8. Furthermore, as proved in [8] and [17],

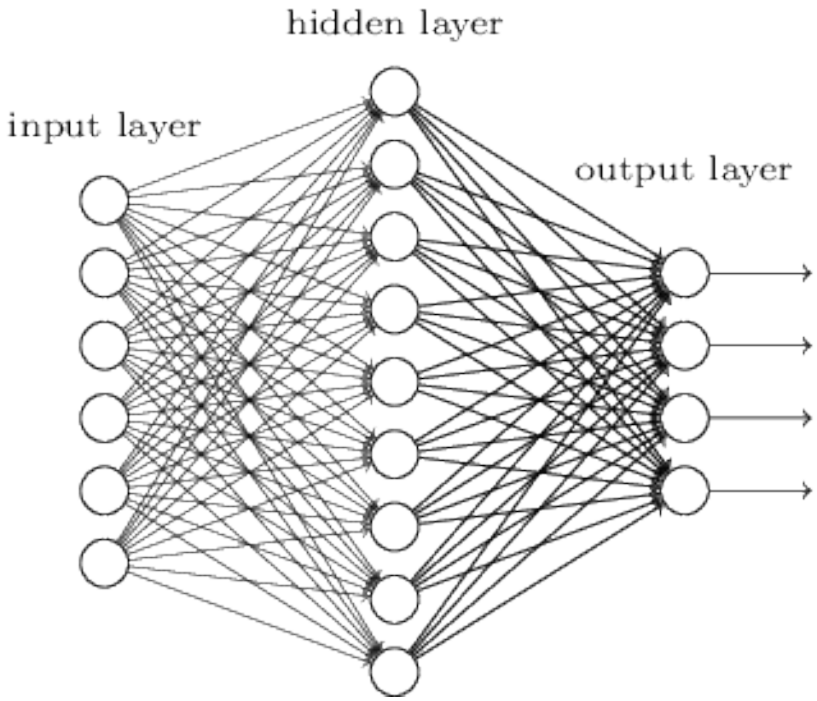

Figure 2.8 *MLP with one hidden layer*

an MLP with a single hidden layer represents a *universal function approximator*. A two-layer perceptron can be written mathematically as:

$$y = f(x) = \varphi(b^{(2)} + W^{(2)}(\varphi(b^{(1)} + W^{(1)}x))) \tag{2.20}$$

where x is a vector of inputs, y a vector of outputs, $W^{(1)}$ and $W^{(2)}$ are weights, $b^{(1)}$ and $b^{(2)}$ are biases and φ is the activation function. In the equation, $\varphi(b^{(1)} + W^{(1)}x)$ forms the hidden layer and the rest of the formula constitutes the output layer.

2.3.3 Training an ANN

ANNs are trained for minimizing the loss function by learning the set of parameters of the model $\theta = \{W^{(*)}, b^{(*)}\}$. The most common approach used for learning the

parameters is called the *stochastic gradient descent* (SGD). The gradients of a loss function are calculated by using the *backpropagation* (BP) algorithm, then they are fed to the SGD method with the aim of updating the weights.

Stochastic Gradient Descent

SGD algorithm updates the set of parameters θ incrementally after each *epoch*. An epoch denotes the number of times all the training examples are used for one forward pass and one backward pass. Unlike in usual gradient descent (GD), where the gradient of the error is computed based on all of the training set, in SGD the gradient of the error is calculated based on a single training sample so that we can stochastically approximate the true error gradient. As a result, we can train an ANN faster by using SGD, because we do not need to compute the gradient for the entire training set for each epoch. In GD, the weights are updated as:

$$w_j := w + \triangle w_j \tag{2.21}$$

$$\triangle w_j = \alpha \sum_{i=1}^{n} (target^{(i)} - output^{(i)}) x_j^{(i)} \tag{2.22}$$

where α is the learning rate. On the other hand, in SGD the weights are also updated as in Equation 2.21, this time $\triangle w$ is defined as:

$$\triangle w_j = \alpha (target^{(i)} - output^{(i)}) x_j^{(i)} \tag{2.23}$$

Backpropogation

When we want to use SGD with multi-layer networks, we need to somehow compute the gradient of the loss function. BP algorithm is the most common method used to overcome this problem. In BP, we basically want to calculate the partial derivatives $\partial L / \partial w$ of the loss function L with respect to some weight w so that we can analyze how fast the loss changes when we change the weights. Let us use mean squared error (MSE) as our cost function. Then, MSE of one output neuron over all n examples is:

$$L = \frac{1}{2} \sum_{j=1}^{n} (t_j - y_j)^2 \tag{2.24}$$

where t is the target label and y is the output of the perceptron. We scale L by $\frac{1}{2}$ for mathematical convenience of Equation 2.30. In order to use SGD, we need to calculate the error gradient.

$$\triangle w_{kj} = -\alpha \frac{\partial L}{\partial w_{kj}} \tag{2.25}$$

where a node in layer k is connected to a node in layer j. We take the negative, because weight changes are in the direction where the error is decreasing. By using chain rule we get:

$$\frac{\partial L}{\partial w_{kj}} = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial x_j} \frac{\partial x_j}{\partial w_{kj}} \tag{2.26}$$

In Equation 2.26, x_j is the weighted sum of the inputs being passed to j^{th} node and $y_j = f(x_j)$ is the output of the activation function. As a result:

$$\frac{\partial x_j}{\partial w_{kj}} = y_k \tag{2.27}$$

Let us consider the sigmoid function as our activation function, then its derivative yields:

$$\frac{df(x)}{dx} = f(x)(1 - f(x)) \tag{2.28}$$

Then, when we plug it in Equation 2.26, we get:

$$\frac{\partial y_j}{\partial x_j} = y_j(1 - y_j) \tag{2.29}$$

Finally, we take the first partial derivative of the remaining part $\frac{\partial L}{\partial yj}$, which is the derivative of Equation 2.24:

$$\frac{\partial L}{\partial y_j} = -(t_j - y_j) \tag{2.30}$$

Overall, putting all things together, we form the algorithm for the output-layer case:

$$\frac{\partial L}{\partial w_{kj}} = -(t_j - y_j) y_j (1 - y_j) y_k \tag{2.31}$$

For the propagation of the error, we can write the chain rule as:

$$\frac{\partial L}{\partial y_j} = \sum \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial x_j} \frac{\partial x_j}{\partial w_{kj}} \tag{2.32}$$

In the Equation 2.32, if we donate the first two partial derivatives as δ_i and the rest is the derivate of the weighted sum of the inputs w_{ji}:

$$\frac{\partial L}{\partial y_j} = -\sum \delta_i w_{ji} \tag{2.33}$$

As a result, we can write the case for the hidden layers:

$$\frac{\partial L}{\partial w_{kj}} = -\sum (\delta_i w_{ji}) y_j (1 - y_j) y_k \tag{2.34}$$

In the end, we can feed the gradient of the calculated error to the SGD algorithm.

3. METHODOLOGY

In this chapter, the details of the implementation steps for the thesis work are described. Throughout this chapter, the following steps will be covered respectively: data preprocessing, feature extraction and ANN architectures.

3.1 Preprocessing

As briefly mentioned in Chapter 2.1, preprocessing is an important task to properly utilize the input signal. In this section, preprocessing for color constancy and also for classification will be briefly discussed.

3.1.1 Preprocessing for Color Constancy

Several non-idealities must be corrected from raw camera sensor input data for the feasibility of the model given in Equation 1.1. Furthermore, since computational color constancy algorithms mentioned in Chapter 2.2.1 based on an analysis of the image contents, it is essential to capture the necessary information of the photographed image scene.

Exposure Control

Exposure control has a significant part in determining how well the information is captured in the raw camera data, because it regulates how light or dark an image appears when captured by a camera. There are several cases that result in loss of information due to bad exposure control, such as quantization of pixel data to zero due to too short exposure time, saturation of pixel data due to overexposure and loss of detail due to motion blur resulting from long exposure time [22]. As a result, we want the best exposed photographed image scene, possibly without loss of information, for efficient and useful color constancy.

Linearity of the Data

Since von Kreis Model assumes that the image data is linear, decent linearization forms another important preprocessing step for color constancy. One of the causes for non-linearity is called *data pedestal*, also known as black level. It refers to the signal level of raw camera data resulting in complete darkness [22]. Another problem is caused by the non-linearity of the dynamic range of the camera. To remove this effect, we omit all saturated pixels from the AWB calculation, because they do not contain the real white points anymore for that image area. White point means that what is the RGB of the prevailing illumination in the linear sensor RGB color space. If the scene has perfect achromatic surface, then it reflects the illumination chromaticity, but the ground truth is defined regardless of the scene contents including perfect achromatic surfaces or not. Hence, allowing saturated image areas (e.g. some parts of bright sky that do not fit in the dynamic range of the sensor) in the calculation will bias the result towards x1.0 white balance gains.

Color Shading

Color shading basically means the lack of color uniformity across the image. Hence, colors from the center of an image may differ from that of the image corners [9]. Thus, it can disturb color appearance and effect the ideal performance of AWB algorithms. As explained in [24] and [23], the effects can be reduced using digital image processing. Both corrected and uncorrected color shading are illustrated in Figure 3.1.

3.1.2 Preprocessing for Classification Task

Apart from preprocessing the data for color constancy, we also used a method for improving the correct classification accuracy of the ANNs. The method which was utilized is called *histogram stretching*. It is an image enhancement technique to improve the contrast by stretching the dynamic range of an image according to a mapping function. Generally, it is defined as:

$$O(x,y) = 255 \cdot \frac{[I(x,y) - MIN]}{[MAX - MIN]} \tag{3.1}$$

Figure 3.1 *Corrected vs. uncorrected color shading [9]*

where $O(x, y)$ is the output image pixel at coordinates x and y, $I(x, y)$ is the input image pixel at coordinates x and y, MIN is the minimum pixel value in the input image and MAX is the maximum pixel value in the input image. This method proved to be useful with Shi-Gehler dataset by increasing the correct classification accuracy up to 5%. Nevertheless, for Intel datasets, it was observed that it did not have a significant effect to compensate for the effort to employ it.

3.2 Feature Extraction

The key thing about the image features is that they will form the input of the neural networks by containing enough information about the scene to allow discriminating between different cases. Nonetheless, they should not have too much irrelevant information, which can distract some algorithms. Most of the features are taken as a reference from the works of [2].

3.2.1 Color Histograms

Color histogram is broadly used in image retrieval problems. Nevertheless, it can also be utilized as a feature for AWB problem as it represents the color distribution of an image so that the system can understand the scene of the image up to a certain point. To that end, RGB color histograms are preferred as a part of the image feature vector. To compute the histogram, RGB color space is quantized into 3x3x3 equal bins and each of the original colors is mapped to a corresponding bin.

3.2.2 YCbCr Color Moments

As mentioned in [2], by transforming an image to the YCbCr color space, the luminance component and the chrominance components can be separated. As in color histograms, YCbCr color moments can be employed to describe the color distribution of an image. Furthermore, as stated in [19], color moments are invariant under photometric changes, which makes them proper feature candidates. To that end, first four color moments is utilized as a part of the image feature vector.

3.2.3 Number of Colors

All of the 4 AWB algorithms used in this thesis take Grey World algorithm as their basis. Thus, the color range gives us a good hint about if the Grey World assumption holds or not. As a consequence, the number of distinct colors in an image is used to represent the color range in the image feature vector.

3.2.4 Cast Indexes

The cast index intends to recognize the existence of a significant cast within the image. In order to calculate the cast indexes, the image is transformed into the YCbCr space, as a very strong cast will show one clear peak within the CbCr plane. The means and variances of the Cb and Cr components (μ_{Cb} and μ_{Cr}, σ^2_{Cb} and σ^2_{Cr}) are used to compute the color equivalence circle center C and its radius r, as well as the two cast indexes D and D_σ:

$$C = (\mu_{Cb}, \mu_{Cr}) \tag{3.2}$$

$$r = \sqrt{\sigma_{Cb}^2 + \sigma_{Cr}^2} \tag{3.3}$$

$$D = C - r \tag{3.4}$$

$$D_\sigma = \frac{D}{r} \tag{3.5}$$

D is a measure of how far the color distribution is from the neutral axis and D_σ quantifies the strength of the cast [2]. However, it should be mentioned that the cast depends a lot on the sensitivity of the used camera sensor. When applied to raw data, the cast depends heavily on the response of the sensor, on top of the illumination and object surface color. So, it will not be invariant with respect to changing properties of camera sensors. Even if normalized before cast calculation, it does not tell whether the cast is due to illumination or object surface reflectance.

3.2.5 Number of Edges

The consistency of the edge-based AWB algorithms could be deteriorated if the scene lacks details and edges. Furthermore, it would also allow separating the impact from edges of different contrast. As a result, the number of edges in an image provides reasonable information for the problem, as also one of the AWB algorithms correlates with the edges. Nonetheless, it is good to mention that the problem with binary edge image is the thresholding. Accumulating the values of edge image (e.g. Sobel without thresholding, or simple high pass filter) into 1 value, and then normalizing it with some value that corresponds to highly textured area, might work more consistently and smoothly, without any steps in the behavior that might come from thresholding. Furthermore, it would allow separating the impact from edges of different contrast.

3.3 ANN Architectures

Two different approaches are experimented to find a better resulting algorithm than GW, GWOC, SOG and WGE, which were presented earlier. Our approaches somehow combine the illuminant estimate RGBs of those 4 AWB algorithms and generate a weighted illuminant estimate. For both methods, we use ANNs consisting of two layers; one hidden and one output layer. Image features generated by using the methods mentioned in Chapter 3.2 contain 43 neighborhood attributes that act as

inputs to the neural networks. For training, *scaled conjugate gradient backpropagation*, which produces generally faster convergence, is used and for loss function, *cross-entropy* method is preferred, as it is more suitable for classification problems. It gives positive output and as the neuron gets better with training, the output gets close to zero. Over all n examples, for a single neuron, it can be written mathematically as:

$$L = -\frac{1}{n}\sum_{x}[y\, ln(t) + (1-y)\, ln(1-t)] \tag{3.6}$$

where t is the output from the neuron and the sum is over all training inputs, x and y is the corresponding desired output. Then, generalizing the cross-entropy method to many-neuron multi-layer networks is a trivial task. Furthermore, for the activation function of the output layer, we use *softmax* function. It produces real valued outputs in the range $[0, 1]$ that add up to 1. As a result, it produces output in the form of a probability distribution to construct a weighted average. For the classification task, we take the position where the intensity is maximal in the output of the softmax function. The function can simply be written as:

$$softmax(x) = \frac{e^{x_i}}{\sum_{j}^{n} e^{x_j}} \tag{3.7}$$

where x_i is an output data and n is the number of total outputs. Furthermore, for model assessment of each method, we prefer 5-fold cross validation. Moreover, one design principle related to system stability, that is also good to keep in mind, is that small change in input should cause only small change in output. If the system only selects one AWB algorithm, it can happen that small change in the image framing can make the system to switch to another AWB algorithm, creating a step change in the colors. In other words, the system should not be impacted by so small image details, otherwise it will be easily impacted by motion blur, small AF error, more blurry optics, etc. error sources.

3.3.1 AWB Algorithm Weighting with Probabilistic Approach

The first method is used to classify images by the success of a color constancy algorithm based on the image features defined earlier. The respective target for each is a 4-element class vector with a 1 in the position of the associated color constancy algorithm, $(1, 0, 0, 0)^T$ for Grey World, $(0, 1, 0, 0)^T$ for Grey World One

Count, $(0,0,1,0)^T$ for Shades of Grey and $(0,0,0,1)^T$ for Weighted Grey Edge. After calculating angular errors for each image and algorithm, for each image, the algorithm with the lowest error rate is classified as the best one, and thereby the target matrix is generated. The network is designed by using the target matrix to train the network in order to produce the correct target classes for AWB algorithms. Several experiments were conducted to determine the optimum number of neurons for the hidden layer and for selecting the best model available for the artificial neural network. It was found out that the model had the best accuracy between bins 50 and 200. Nonetheless, the maximum accuracy is achieved by using 50 hidden neurons. In the end, for Shi-Gehler dataset, 48% correct classification accuracy can be achieved for the 4 class problem, where the random probability of selecting the correct class is 25%. Furthermore, by using softmax transfer function in the output layer, probability distribution over each AWB algorithm can be obtained. As a result, AWB algorithm weighting becomes feasible, since the better the classification accuracy the better the probability distribution over each AWB algorithm. In the end, we can generate a fifth algorithm NN_AWB_1, which is a linear combination of the white points of the 4 AWB algorithms. The weighting (linear combination) is based on the outcome of the NN for a given frame. To make it more concrete, let us assume that for one example frame the algorithms give white points $[R,G,B]$ and NN gives weights for each algorithm as below in Table 3.1. We get one white

Table 3.1 *Example case of 1st method*

		GW	GWOC	SoG	WGE
WP	Red	120	130	140	150
	Green	150	160	170	180
	Blue	140	130	120	110
NN_result	Probabilities	0.2	0.3	0.1	0.4

point for the NN_AWB_1 ([137,167,123] in this example case). Comparing that to the ground truth white point, we get the angular error for NN_AWB_1, which can be compared against GW, GWOC etc. results. This is done for all images in the database.

3.3.2 AWB Algorithm Weighting with Combining Separate Neural Networks Approach

The second method involves having separate neural networks for each AWB algorithm, with output nodes associated with different angular error ranges shown in Table 3.2. We think that this method could work better due to having more consistent correlation between image contents and the AWB algorithm performance. In the first method there can be discontinuities coming from the fact that if some other AWB algorithm is just slightly better, then that other AWB gets selected, even though the image contents could be almost the same as in some other image in which the order of the AWB algorithms is the opposite. Hence, instead of indicating the right AWB, in this method, ANNs indicate the likelihood of success for each AWB, but we need multiple ANNs instead of the 1 ANN that we have in the 1st approach. In this case, the respective target for each is a 5-element class vector with

Table 3.2 *Possible error classes and ranges*

Error class (ec)	1	2	3	4	5
Angular error	[0, 1)	[1, 3)	[3, 6)	[6, 9)	[9 , ∞)

a 1 in the position of the associated error class, just as in the 1st approach. For a given image, we know the error class *ec* for each AWB algorithm, based on the NN that is associated with each AWB algorithm. Similarly, we have the white points from each AWB algorithm. To get neural network white point (NN_AWB_2) for a given image, we select the white point(s) that correspond to the lowest *ec*, and average them (if there is only one AWB algorithm in the lowest error class then no averaging needed). To that end, we train 4 different neural networks; one for each AWB algorithm. For Shi-Gehler dataset, general model accuracies for correct classification are 35% for both GW and GWOC, 40% for SOG and 43% for WGE, where the random probability of selecting the correct class is 20%.

4. EXPERIMENTAL SETUP AND RESULTS

In this chapter, experimental setup and results are discussed in detail. The angular error results are presented in tables. The implementation platform for this thesis work was MATLAB. MATLAB's Neural Network Toolbox is used in the implementation.

4.1 Datasets

The performance of AWB algorithms were tested on several datasets, one is reprocessed version of Shi-Gehler RAW dataset, which is considered as one of the benchmarks and publicly available [27] and others are Intel's own confidential datasets having images from different camera sensors under different illumination conditions.

The Shi-Gehler RAW dataset contains 568 indoor and outdoor images taken using Canon 5D and Canon 1D digital cameras with all settings in auto mode. The dataset was originally provided by [13]. Each image contains a MacBeth colorchecker for reference, which is illustrated in Figure 4.1. By using a MacBeth colorchecker we can calculate the ground truth white points, because gray patches of the colorchecker directly indicate the white points, since they reflect the illumination chromaticity without altering it.

For this thesis, we only used Canon 5D images, whose spatial resolution is 2193 x 1460, because we did not want to mix images from different sensors. Moreover, they have the black level of 129, which needs to be subtracted before any further process. A set of example images of Shi-Gehler dataset are illustrated in Figure 4.2.

4.2 Error Measure

In this thesis, angular error is used to measure the error. The angle between the RGB triplet of the ground truth illuminant e and the RGB triplet of the estimated

illuminant $\hat{e}$ can be defined as:

$$e_{Ang} = arccos(\frac{e^T \hat{e}}{||e||\ ||\hat{e}||}) \tag{4.1}$$

where $||.||$ is the Euclidean norm. It is also good to note that the ground truth RGB triplet is sensor-specific. In Section 4.3, we present the minimum, median, mean and maximum of obtained angular errors.

Figure 4.1 *MacBeth colorchecker*
(Source: https://luminous-landscape.com/sinar-hy6)

4.3 Results

For the Shi-Gehler dataset the results of each approach are show in Table 4.1. The results look promising for NN_AWB_1 method, which was the results of AWB algorithm weighting with probabilistic approach, as the median error is clearly better than GWOC. It is safe to assume that there are a bunch of high error cases for NN_AWB_1 that push the mean error a bit higher than the one of GWOC. On the other hand, it can also be observed from the Table 4.1 that NN_AWB_2 is clearly better than all other algorithms, including NN_AWB_1. It only fails to beat GWOC on maximum error results.

Figure 4.2 *Example images of Shi-Gehler dataset*

Furthermore, overall error distribution of NN_AWB_2 is illustrated in Figure 4.3 and the most problematic image, which contains large colored areas, is shown in Figure 4.4. It can be analyzed that maximum error image is one of the several isolated cases and as also the mean value suggests, most of the images are within 5 degree error range. As a result, NN_AWB_2 method proves to be very promising even with average classification accuracy.

Table 4.1 *Angular error results for the Shi-Gehler dataset*

Algorithm	Min	Mean	Median	Max
GW	0.03	4.59	3.54	19.17
GWOC	0.05	3.68	2.91	**16.44**
SoG	0.08	5.83	3.09	36.18
WGE	0.04	5.20	2.04	33.39
NN_AWB_1	0.05	3.97	1.94	21.39
NN_AWB_2	**0.02**	**3.41**	**1.72**	24.41

In Table 4.2, Table 4.3, Table 4.4, Table 4.5, Table 4.6 and Table 4.7, results of the Intel datasets are presented. Each dataset has images from different camera sensors and scenes under various illuminations. There are 6 databases; from intel_1 to intel_6 and the numbers of images in each database are 69, 280, 196, 188, 72 and 78 respectively. Furthermore, as it can observed from the tables SoG algorithm yields

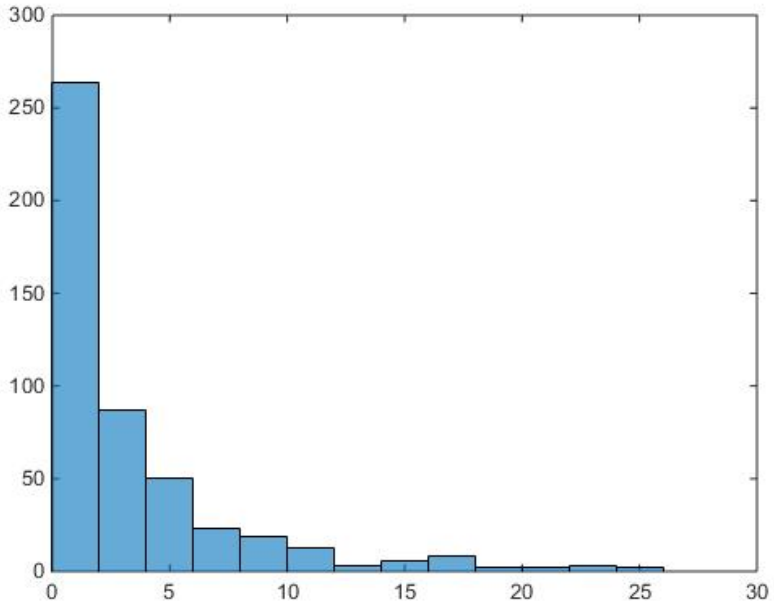

Figure 4.3 *Histogram of the angular errors of 2nd method for Shi-Gehler dataset*

Figure 4.4 *The most problematic image*

very high error rate. This is because of the L norm we used. Nonetheless, instead of finding the optimal norm, we decided to keep it, because the only difference to GW is that brighter values get higher weight, and it seems that it does not suit well for

those databases, but the ANN should be able to pick the images in which is does work (e.g. the brightest objects are some white objects that reflect the illumination chromaticity). If the L is decreased, it would just become more similar to GW, and that does not really benefit the system.

As tables suggest, generally, NN_AWB_2 method significantly outperforms those 4 AWB approaches in terms of mean and median error, and even for some cases it also outperforms in terms of minimum and maximum error.

4.3.1 Camera Sensor Invariance and Error Quantification

In practice we do not want to train the ANNs again when new camera module type is used, because it is time consuming, expensive, and error prone to capture a new training database that has good enough coverage. In this part we quantify how much the performance of the ANNs degrade when the test sensor is different than the training sensor. The challenge here is that we do not have exactly the same raw image contents captured with multiple cameras. Nevertheless, in the end of this part we should have a good idea of how much degradation happens, both in terms of classification accuracy and NN_AWB_2 angular error. To that end, we trained one ANN with one Intel database and tested on 5 others, and repeated so that all databases get to be the training database in turns. AWB results are presented in Table 4.8, Table 4.9, Table 4.10, Table 4.11, Table 4.12 and Table 4.13. Accuracies of the ANNs on providing the correct error are presented in Table 4.14, Table 4.15, Table 4.16, Table 4.17, Table 4.18 and Table 4.19. In the accuracy report, the values are given in tuples as (Org/Inv) which means, Org value is the original accuracy of an ANN tested by the same database, whereas Inv shows the accuracy when tested by another database. The results clearly show the need for additional work to gain invariance against changes in camera module properties. There is also impact from the image contents, which is quite different between the databases. As a result, we also experimented with 2 different camera sensors, Intel Saltbay's and Nikon D810, to also see the same metrics for a case when the image contents and framing is almost the same, but camera properties are different. Then the impact would only be from the change in camera properties. If the system instead has more continuous weights for all algorithms, it is possible at least in principle to produce more continuous changes when the framing changes. AWB results of this experiment are presented in Table 4.20 and Table 4.21, and accuracies of the ANNs on providing the correct error are presented in Table 4.22. However, from the results

it can be analyzed that even in that conditions, the results clearly show the need for additional work to gain invariance against changes in camera module properties.

Table 4.2 *Angular error results for Intel_ 1 dataset*

Algorithm	Min	Mean	Median	Max
GW	**0.16**	**4.98**	**2.58**	**17.76**
GWOC	0.19	6.36	4.82	21.66
SoG	7.36	12.18	11.46	23.18
WGE	0.76	5.54	4.28	22.84
NN_AWB_2	0.20	5.20	2.90	**17.76**

Table 4.3 *Angular error results for Intel_ 2 dataset*

Algorithm	Min	Mean	Median	Max
GW	**0.10**	4.74	4.62	**12.29**
GWOC	0.27	6.39	5.67	40.59
SoG	8.98	10.49	9.90	21.68
WGE	0.20	4.46	3.88	17.37
NN_AWB_2	0.19	**4.03**	**3.66**	15.47

Table 4.4 *Angular error results for Intel_ 3 dataset*

Algorithm	Min	Mean	Median	Max
GW	0.35	4.57	3.43	**19.95**
GWOC	**0.29**	8.25	6.48	39.13
SoG	8.39	11.70	9.83	21.31
WGE	0.34	4.63	3.28	22.03
NN_AWB_2	0.34	**3.73**	**2.62**	22.38

Table 4.5 *Angular error results for Intel_4 dataset*

Algorithm	Min	Mean	Median	Max
GW	0.04	**3.74**	3.24	**10.80**
GWOC	0.11	8.96	6.16	44.06
SoG	13.05	15.88	14.34	24.52
WGE	**0.04**	6.00	4.67	24.12
NN_AWB_2	**0.04**	3.98	**2.88**	33.01

Table 4.6 *Angular error results for Intel_5 dataset*

Algorithm	Min	Mean	Median	Max
GW	**0.09**	9.87	9.71	20.44
GWOC	0.54	10.01	9.87	21.21
SoG	14.89	15.43	15.51	**16.94**
WGE	0.97	7.69	**6.72**	22.19
NN_AWB_2	**0.09**	**7.40**	6.82	18.85

Table 4.7 *Angular error results for Intel_6 dataset*

Algorithm	Min	Mean	Median	Max
GW	**0.22**	6.41	5.83	19.47
GWOC	0.23	8.53	7.73	36.87
SoG	13.02	16.27	15.44	20.23
WGE	1.15	6.75	6.49	**16.82**
NN_AWB_2	0.23	**5.97**	**4.90**	21.39

Table 4.8 *Angular error results for Intel_ 1 dataset for error quantification*

Algorithm	Min	Mean	Median	Max
GW	**0.16**	4.98	**2.58**	17.76
GWOC	0.19	6.36	4.82	21.66
SoG	7.36	12.18	11.46	23.18
WGE	0.76	5.54	4.28	22.84
NN_AWB_intel_*	0.20	5.20	2.90	17.76
NN_AWB_intel_2	0.30	4.63	2.64	21.66
NN_AWB_intel_3	0.30	**4.37**	2.59	**15.12**
NN_AWB_intel_4	**0.16**	5.00	2.87	16.96
NN_AWB_intel_5	**0.16**	5.19	3.48	19.47
NN_AWB_intel_6	0.19	5.37	3.91	17.39

Table 4.9 *Angular error results for Intel_ 2 dataset for error quantification*

Algorithm	Min	Mean	Median	Max
GW	**0.10**	4.74	4.62	**12.29**
GWOC	0.27	6.39	5.67	40.59
SoG	8.98	10.49	9.90	21.68
WGE	0.20	4.46	3.89	17.37
NN_AWB_intel_*	0.19	**4.03**	**3.66**	15.47
NN_AWB_intel_1	0.22	4.59	4.23	19.28
NN_AWB_intel_3	0.21	4.59	4.11	19.28
NN_AWB_intel_4	0.16	4.41	3.84	19.28
NN_AWB_intel_5	**0.10**	4.59	3.89	28.19
NN_AWB_intel_6	**0.10**	4.89	4.31	29.92

Table 4.10 *Angular error results for Intel_3 dataset for error quantification*

Algorithm	Min	Mean	Median	Max
GW	0.35	4.57	3.43	19.95
GWOC	0.29	8.25	6.48	39.13
SoG	8.39	11.70	9.83	21.31
WGE	0.34	4.63	3.28	22.03
NN_AWB_intel_*	0.34	**3.73**	**2.62**	22.38
NN_AWB_intel_2	**0.10**	4.63	3.84	22.38
NN_AWB_intel_1	0.35	5.37	4.42	24.96
NN_AWB_intel_4	0.36	4.57	3.48	**18.06**
NN_AWB_intel_5	0.37	5.40	3.58	25.54
NN_AWB_intel_6	0.17	5.10	3.77	39.13

Table 4.11 *Angular error results for Intel_4 dataset for error quantification*

Algorithm	Min	Mean	Median	Max
GW	**0.04**	**3.74**	3.24	**10.80**
GWOC	0.11	8.96	6.16	44.06
SoG	13.05	15.88	14.34	24.52
WGE	**0.04**	6.00	4.67	24.12
NN_AWB_intel_*	**0.04**	3.98	**2.88**	33.01
NN_AWB_intel_2	0.19	4.98	4.02	19.78
NN_AWB_intel_3	0.15	5.09	3.95	21.46
NN_AWB_intel_1	**0.04**	5.34	4.26	39.52
NN_AWB_intel_5	0.19	6.26	4.69	38.94
NN_AWB_intel_6	0.11	5.91	4.49	31.62

Table 4.12 *Angular error results for Intel_5 dataset for error quantification*

Algorithm	Min	Mean	Median	Max
GW	**0.09**	9.87	9.71	20.44
GWOC	0.54	10.01	9.87	21.21
SoG	14.89	15.43	15.51	**16.94**
WGE	0.97	7.69	**6.72**	22.19
NN_AWB_intel_*	**0.09**	**7.40**	6.82	18.85
NN_AWB_intel_2	0.84	8.98	8.52	22.19
NN_AWB_intel_3	0.97	8.43	7.90	22.19
NN_AWB_intel_4	0.22	8.40	8.03	21.21
NN_AWB_intel_1	0.54	9.64	9.14	22.19
NN_AWB_intel_6	**0.09**	8.86	9.02	18.17

Table 4.13 *Angular error results for Intel_6 dataset for error quantification*

Algorithm	Min	Mean	Median	Max
GW	**0.22**	6.41	5.83	19.47
GWOC	0.23	8.53	7.73	36.87
SoG	13.02	16.27	15.44	20.23
WGE	1.15	6.75	6.49	**16.82**
NN_AWB_intel_*	0.23	**5.97**	**4.90**	21.39
NN_AWB_intel_2	0.32	6.26	5.75	19.47
NN_AWB_intel_3	0.32	6.65	6.23	21.39
NN_AWB_intel_4	0.36	6.26	5.40	19.72
NN_AWB_intel_5	1.14	7.66	7.65	17.57
NN_AWB_intel_1	**0.22**	7.34	6.77	21.03

Table 4.14 *Accuracy report when intel_1 is used for training & others testing*

Algorithm	intel_2	intel_3	intel_4	intel_5	intel_6
GW	43.5 / 24.6	40.0 / 27.6	32.8 / 24.2	60.8 / 18.1	28.2 / 19.2
GWOC	31.8 / 25.4	33.7 / 23.0	36.0 / 20.3	53.1 / 20.7	24.2 / 24.3
SoG	99.5 / 94.0	96.6 / 92.1	100.0 / 88.7	100.0 / 59.7	100.0 / 81.2
WGE	41.7 / 25.2	33.7 / 24.8	28.8 / 22.5	36.7 / 18.9	23.8 / 24.1

Table 4.15 *Accuracy report when intel_2 is used for training & others testing*

Algorithm	intel_1	intel_3	intel_4	intel_5	intel_6
GW	34.6 / 28.2	40.0 / 33.6	32.8 / 24.7	60.8 / 23.2	28.2 / 23.9
GWOC	27.7 / 28.0	33.7 / 29.9	36.0 / 27.8	53.1 / 20.0	24.2 / 21.8
SoG	91.7 / 87.6	96.6 / 95.1	100.0 / 93.9	100.0 / 68.8	100.0 / 88.5
WGE	30.1 / 29.6	33.7 / 26.7	28.8 / 24.6	36.7 / 25.0	23.8 / 23.0

Table 4.16 *Accuracy report when intel_3 is used for training & others testing*

Algorithm	intel_2	intel_1	intel_4	intel_5	intel_6
GW	43.5 / 29.2	34.6 / 25.6	32.8 / 22.6	60.8 / 22.1	28.2 / 24.3
GWOC	31.8 / 28.1	27.7 / 23.5	36.0 / 27.4	53.1 / 21.8	24.2 / 26.0
SoG	99.5 / 95.5	91.7 / 88.1	100.0 / 89.0	100.0 / 70.6	100.0 / 87.4
WGE	41.7 / 24.1	30.1 / 27.5	28.8 / 25.3	36.7 / 24.4	23.8 /25.3

Table 4.17 *Accuracy report when intel_4 is used for training & others testing*

Algorithm	intel_2	intel_3	intel_1	intel_5	intel_6
GW	43.5 / 28.7	40.0 / 25.2	34.6 / 24.6	60.8 / 14.2	28.2 / 21.3
GWOC	31.8 / 27.0	33.7 / 27.8	27.7 / 23.1	53.1 / 12.7	24.2 / 25.2
SoG	99.5 / 99.6	96.6 / 97.5	91.7 / 95.7	100.0 / 100.0	100.0 / 100.0
WGE	41.7 / 23.1	33.7 / 24.9	30.1 / 25.9	36.7 / 26.0	23.8 /27.8

Table 4.18 *Accuracy report when intel_5 is used for training & others testing*

Algorithm	intel_2	intel_3	intel_4	intel_1	intel_6
GW	43.5 / 16.4	40.0 / 15.8	32.8 / 16.6	34.6 / 16.9	28.2 / 24.3
GWOC	31.8 / 20.3	33.7 / 21.7	36.0 / 20.6	27.7 / 21.1	24.2 / 25.2
SoG	99.5 / 99.6	96.6 / 97.5	100.0 / 100.0	91.7 / 95.7	100.0 / 100.0
WGE	41.7 / 21.6	33.7 / 20.2	28.8 / 22.4	30.1 / 19.4	23.8 / 23.3

Table 4.19 *Accuracy report when intel_6 is used for training & others testing*

Algorithm	intel_2	intel_3	intel_4	intel_1	intel_6
GW	43.5 / 24.7	40.0 / 26.0	32.8 / 20.9	60.8 / 28.6	34.6 / 20.2
GWOC	31.8 / 21.8	33.7 / 24.0	36.0 / 22.4	53.1 / 26.6	27.7 / 23.0
SoG	99.5 / 99.6	96.6 / 97.5	100.0 / 100.0	100.0 / 100.0	91.7 / 95.7
WGE	41.7 / 25.5	33.7 / 24.7	28.8 / 25.0	36.7 / 25.2	30.1 / 24.9

Table 4.20 *Angular error results for Saltbay dataset*

Algorithm	Min	Mean	Median	Max
GW	**0.07**	4.43	4.38	19.68
GWOC	0.44	4.10	3.43	13.70
SoG	15.49	16.95	17.08	18.21
WGE	0.16	1.85	1.47	**6.98**
NN_AWB_2	0.16	**1.78**	**1.45**	7.37

Table 4.21 *Angular error results for Nikon D810 dataset*

Algorithm	Min	Mean	Median	Max
GW	**0.09**	6.28	5.80	27.01
GWOC	6.42	25.60	25.97	48.73
SoG	15.22	16.12	16.14	16.98
WGE	0.20	3.22	2.54	**10.43**
NN_AWB_2	**0.09**	**3.18**	**2.26**	**10.43**

Table 4.22 *Accuracy report for Saltbay vs. Nikon D810*

Algorithm	Saltbay	Nikon D810
GW	40.2 / 19.4	33.9 / 20.5
GWOC	36.9 / 22.7	93.1 / 7.9
SoG	100.0 / 100.0	100.0 / 100.0
WGE	50.4 / 26.4	42.3 / 18.5

5. CONCLUSIONS

In this thesis, we have studied 2 different ANN architectures, namely probabilistic approach and combination approach, to estimate of the color of the light source of a target image. The motivation was to generate a new AWB algorithm by weighting some of the existing AWB algorithms. 4 AWB algorithms were selected to use as a reference, namely grey world, grey world one count, shades of grey and weighted grey edge. Furthermore, we have analyzed camera sensor invariance by quantifying how much the performance of the ANNs degrade when the test sensor is different than the training sensor.

We used ANNs for the task, because in recent studies the problem has been approached with machine learning techniques quite often and they have been proved to be very useful. Furthermore, one of the major advantages of using ANNs is that their ability to generalize so that they can be used for unseen data to make accurate predictions. Furthermore, neural networks are simple to implement and easy to deploy on embedded systems that are used in real life applications such as smart phones and digital cameras.

For computational color constancy, preprocessing is an important task. Several non-idealities must be corrected from raw camera sensor input data for them to be feasible. The most important things to consider for preprocessing step is exposure control, linearity of the data and color shading. One needs to take care of non-idealities before applying color constancy to an image.

To use as inputs to the ANNs, we extracted several features from the images. The key thing about the image features is that they should contain enough information about a given scene to allow discriminating between different cases. To that end, we used color histograms, YCbCr color moments, number of colors, cast indexes and number of edges to form the 43 neighborhood attributes to use as inputs.

Two different approaches were experimented to find a better resulting algorithm than those 4 AWB algorithms presented earlier. Both approaches somehow combine the illuminant estimate RGBs of those 4 AWB algorithms and generate a weighted illuminant estimate. The first method was used to classify images by the success of a color constancy algorithm based on the image features defined earlier, whereas the second method involves having separate neural networks for each AWB algorithm, with output nodes associated with different angular error ranges. For both methods, we used ANNs consisting of two layers; one hidden and one output layer. For training, scaled conjugate gradient backpropagation was used and for loss function, cross-entropy method was preferred. For the output layer, instead of traditional sigmoid or tanh activation functions, softmax activation function were preferred to produce output in the form of a probability distribution to construct a weighted average, especially for the 1st method.

Approaches were trained and tested on 1 publicly available dataset, Shi-Gehler, and several confidential Intel datasets in MATLAB environment. We evaluated the performance of our approaches based on 5-fold cross validation error rate. The experimental results proved that the first approach is better than the existing approaches in terms of median error, while the second method, which is better also from system design point of view, is superior to all others including the first approach in terms of mean and median error.

We also analyzed camera sensor invariance and error quantification to see how much the performance of the ANNs degrade when the test sensor is different than the training sensor. To that end, we used 6 Intel datasets to train one ANN with one database and tested on others, and repeated so that all databases get to be the training database in turns. Furthermore, we also experimented with 2 different camera sensors to also see the same metrics for a case when the image contents and framing is almost the same, but camera properties are different. To that end, we provided angular error reports as well as correct classification accuracies of the experiments. The results clearly show the need for additional work to gain invariance against changes in camera module properties.

To address that problem, if one can improve the neural networks' classification accuracy with a more descriptive feature set or with a better model, then the angular error rates can get better. Similarly, better camera sensor invariance can be achieved. For instance, convolutional neural networks (CNNs) can be implemented. Using

CNNs for AWB task is a quite new and popular idea. For instance, in [3], [4] and [32], they use CNNs for illuminant estimation, which can be investigated for further research.

BIBLIOGRAPHY

[1] V. Agarwal, A. V. Gribok, A. Koschan, and M. A. Abidi, "Estimating illumination chromaticity via kernel regression," *2006 IEEE International Conference on Image Processing*, pp. 981–984, Oct. 2006.

[2] S. Bianco, G. Ciocca, C. Cusano, and R. Schettini, "Automatic color constancy algorithm selection and combination," *Pattern Recognition*, pp. 695–705, March 2010.

[3] S. Bianco, C. Cusano, , and R. Schettini, "Color constancy using cnns," *Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2015.

[4] ——, "Single and multiple illuminant estimation using convolutional neural networks," Dec. 2015. [Online]. Available: http://128.84.21.199/pdf/1508.00998.pdf

[5] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, 2006.

[6] G. Buchsbaum, "A spatial processor model for object colour perception," *Journal of the Franklin Institute*, pp. 1–26, July 1980.

[7] V. C. Cardei, B. Funt, and K.Barnard, "Estimating the scene illumination chromaticity by using a neural network," *Journal of the Optical Society of America A*, pp. 2374–2386, Dec. 2002.

[8] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, pp. 303–314, Dec. 1989.

[9] "Dxo autocls - color uniformity," DxO Labs, 2016. [Online]. Available: http://www.dxo.com/us/embedded-imaging/image-signal-processor-isp/dxo-autocls

[10] G. D. Finlayson and E. Trezzi, "Shades of gray and colour constancy," *Color and Imaging Conference*, pp. 37–41, Jan. 2004.

[11] M. A. Fischler and O. Firschein, *Readings in computer vision : issues, problems, principles, and paradigms.* M. Kaufmann Publishers Inc, 1987.

[12] D. A. Forsyth, "A novel algorithm for color constancy," *International Journal of Computer Vision*, pp. 5–36, Aug. 1990.

[13] P. Gehler, C. Rother, A. Blake, T. Minka, and T. Sharp, "Bayesian color constancy revisited," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2008.

[14] A. Gijsenij, T. Gevers, and J. van de Weijer, "Computational color constancy: Survey and experiments," *IEEE Transactions on Image Processing*, pp. 2475–2489, Sept. 2011.

[15] ——, "Improving color constancy by photometric edge weighting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 918–929, Oct. 2011.

[16] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice Hall, 1998.

[17] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, pp. 251–257, 1991.

[18] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[19] F. Mindru, T. Tuytelaars, L. van Gool, and T. Moons, "Moment invariants for recognition under changing viewpoint and illumination," *Computer Vision and Image Understanding - Special Issue: Colour for Image Indexing and Retrieval*, pp. 3–27, April–June 2004.

[20] T. Mitchell, *Machine Learning.* McGraw Hill, 1997.

[21] A. Moore, J. Allman, and R. M. Goodman, "A real-time neural system for color constancy," *IEEE Transactions on Neural Networks*, pp. 237–247, Mar. 1991.

[22] J. T. Nikkanen, "Computational color constancy in mobile imaging," Ph.D. dissertation, Tampere University of Technology, 2013.

[23] J. T. Nikkanen and O. Kalevo, "Method and system for vignetting elimination in digital image," International Patent Application PCT/IB2005/003 057, 2007.

[24] ——, "Method, apparatus, imaging module and program for improving image quality in a digital imaging devices," Patent US 7 548 262, 2009.

[25] S. J. D. Prince, *Computer vision: models, learning and inference.* Cambridge University Press, 2012.

[26] F. Rosenblatt, *Principles of neurodynamics; perceptrons and the theory of brain mechanisms*. Spartan Books, 1962.

[27] L. Shi and B. Funt, "Re-processed version of the gehler color constancy dataset of 568 images." [Online]. Available: http://www.cs.sfu.ca/~colour/data

[28] R. Stanikunas and H. Vaitkevicius, "Neural network for color constancy," *Informatica*, pp. 219–232, Feb. 2000.

[29] J. van de Weijer and T. Gevers, "Color constancy based on the grey-edge hypothesis," *ICIP 2005. IEEE International Conference on Image Processing*, pp. II–722–5, Sept. 2005.

[30] A. R. Webb and K. D. Copsey, *Statistical Pattern Recognition*, 3rd ed. Wiley, 2011.

[31] W. Xiong and B. Funt, "Estimating illumination chromaticity via support vector regression," *Journal of Imaging Science and Technology*, pp. 341–348, Aug. 2006.

[32] H. Yuzuguzel, "Learning colour constancy using convolutional neural networks," Master's thesis, Tampere University of Technology, Finland, 2015.